Lessen Java: Reeks 3

David Blinder
Jan G. Cornelis

Vraag 0: Gebruik van de Debugger

Syntax errors: fouten door verkeerd gebruik van Java-commandos → code zal niet compileren. Locatie van de fout is bijna altijd duidelijk.

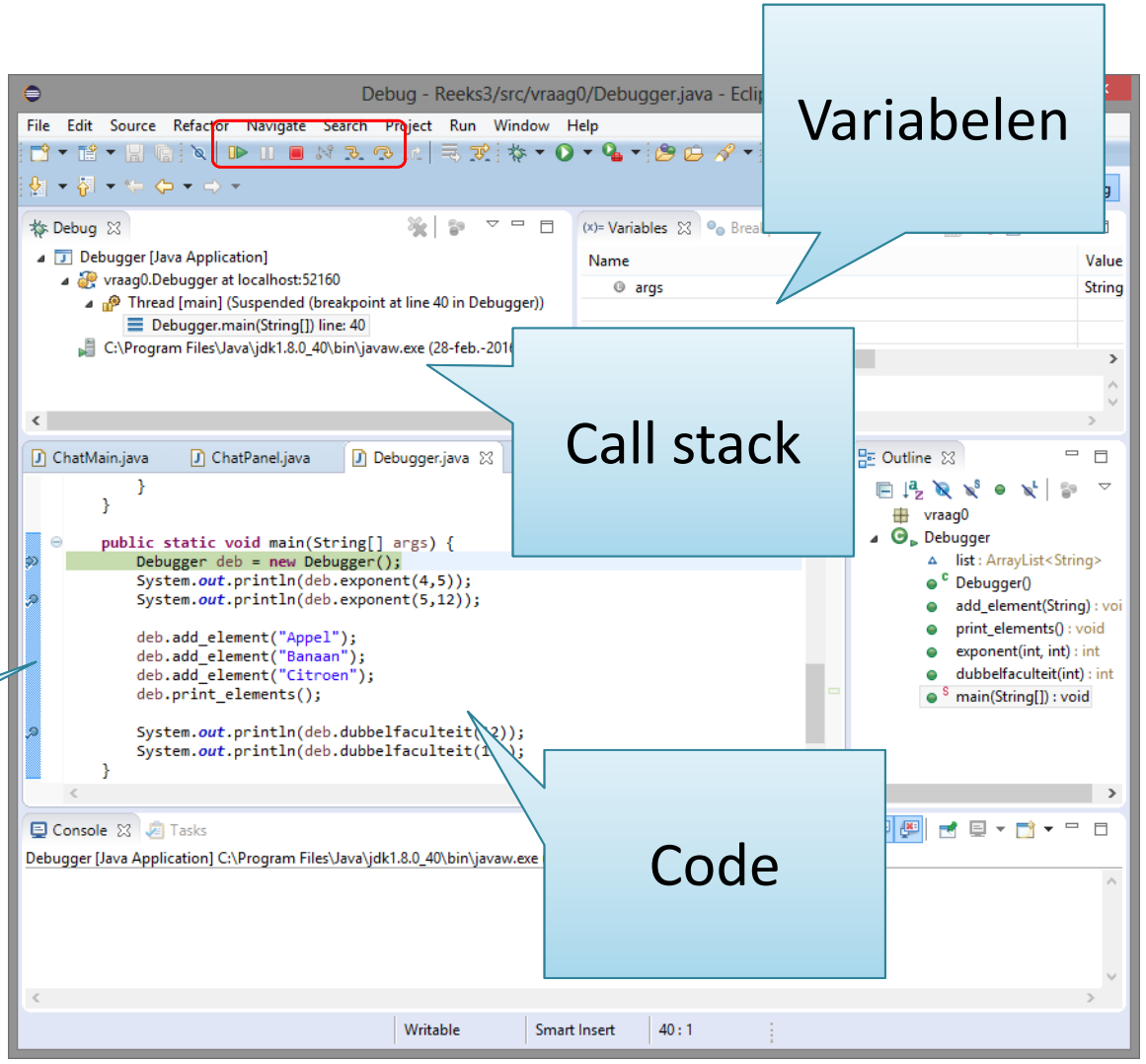
Runtime errors: code is syntactisch correct, maar niet functioneel correct. Mogelijke gevolgen:

- Onmogelijke operatie: bv. deling door nul, toegang tot onbestaande geheugenlocatie.
- Oneindige lus: bv. while lus die nooit eindigt
- Foute resultaten: foute implementatie van een formule

Voor het oplossen van runtime errors kun je gebruik maken van een tool om de bugs te detecteren: de **debugger**

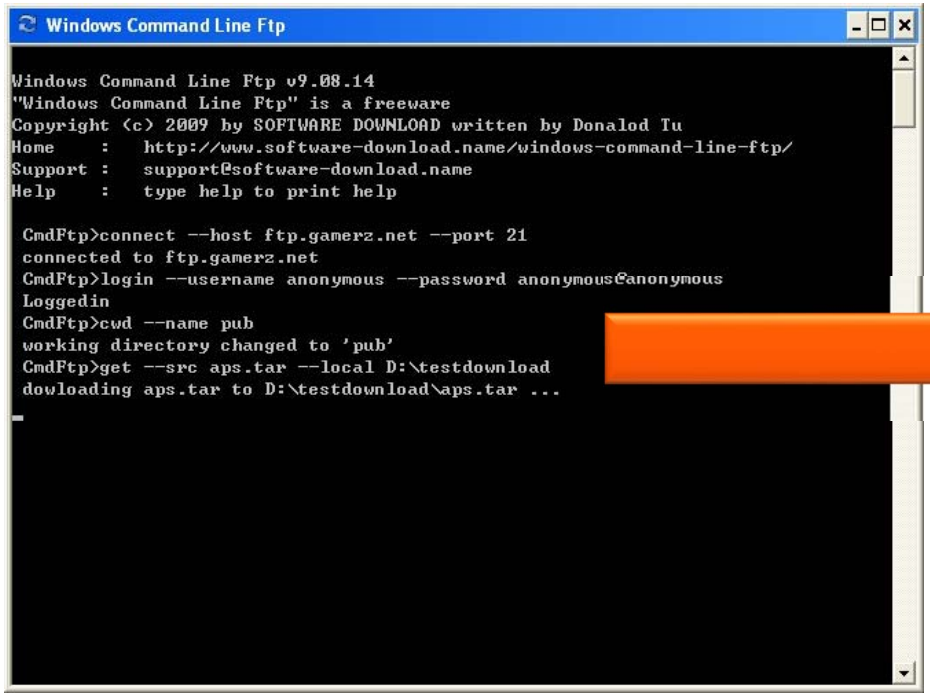
Vraag 0: Debugger

1. Naar volgend breakpoint
2. Stop debugging
3. Stap in methode
4. Volgende lijn

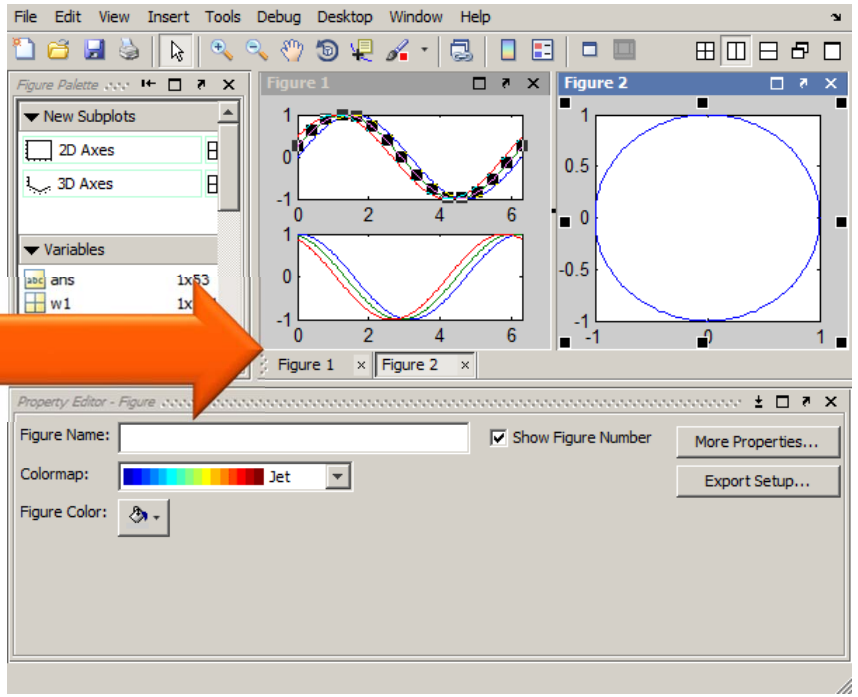


Werken met GUIs

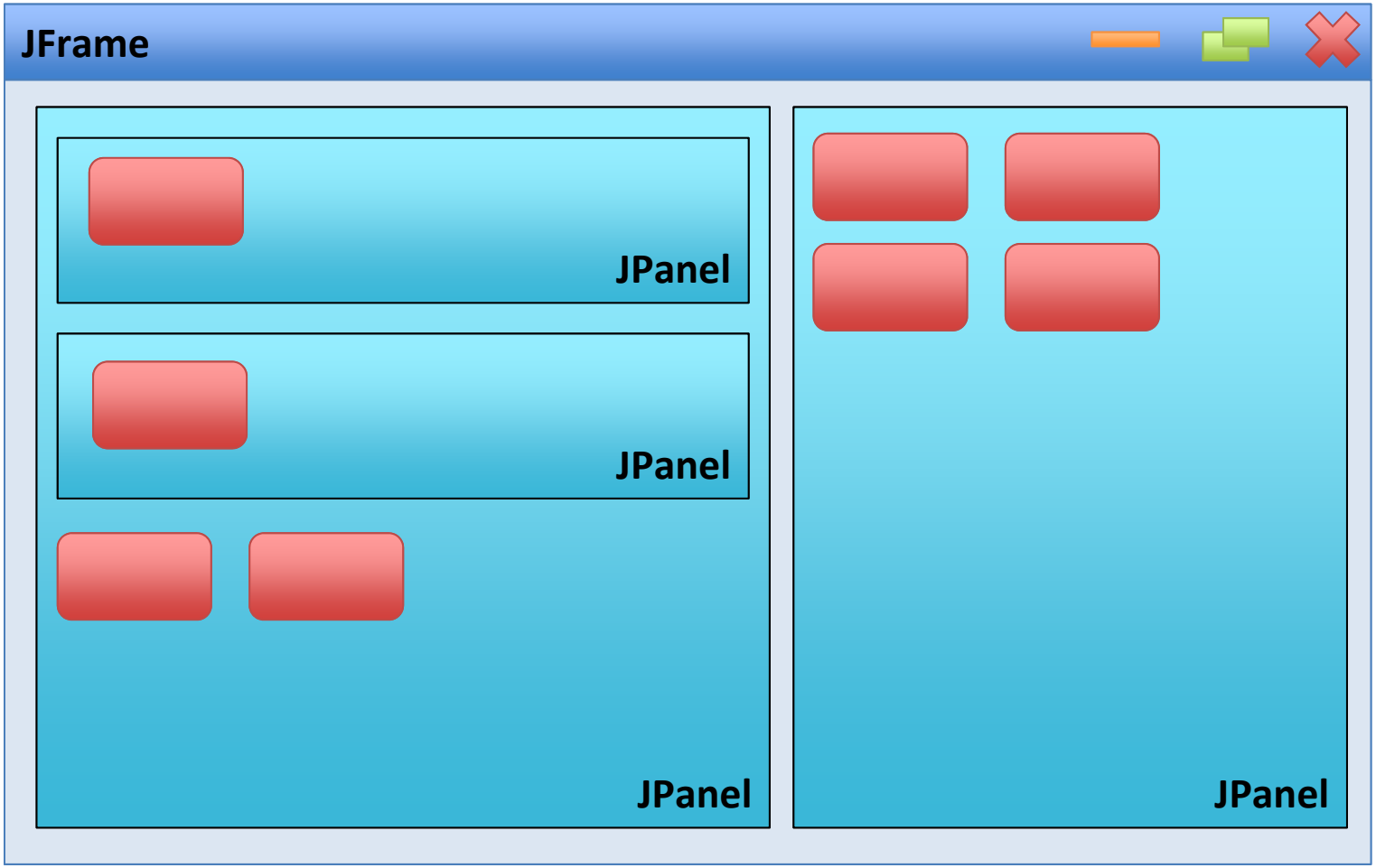
Command line



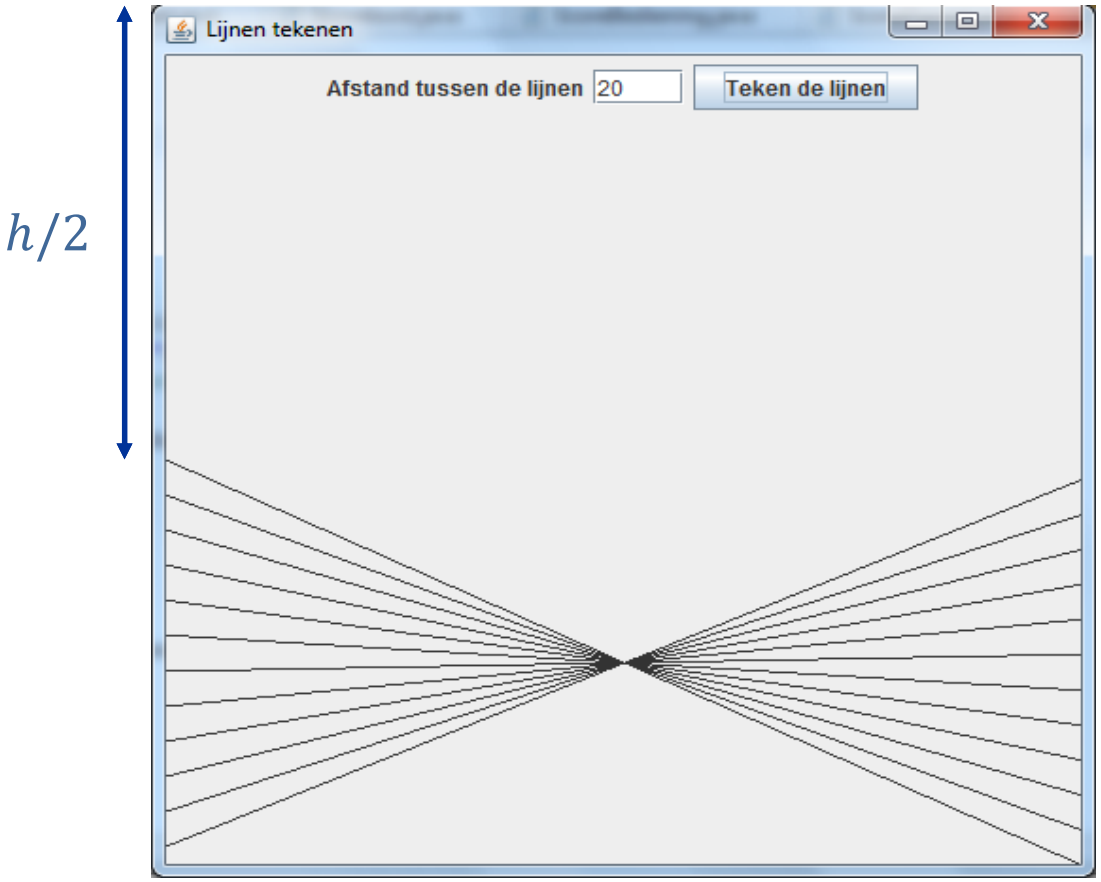
Interactieve vensters



Java “swing” library



Vraag 1: Lijnwaaier



- Teken een lijnwaaier met N lijnen, ingegeven door de gebruiker.
- Maak nieuwe klasse LijnPaneel (JPanel)

Venster aanmaken

```
import java.awt.event.*;
import javax.swing.*;

public class LijnPaneel extends JPanel implements ActionListener
{
…

        public static void main(String[] args)
        {
                JFrame f = new JFrame();
                f.setSize(500,500);
                f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                f.setTitle("Lijnen tekenen");
                f.setLocation(100, 100); //standaard in de hoek van het scherm
                JPanel hoofdpaneel = new LijnPaneel();
                f.add(hoofdpaneel);
                f.setVisible(true);
        }
}
```

Lijnwaaier


JTextField
Lijnen tekenen
Afstand tussen de lijnen
20
Teken de lijnen
JLabel
JButton

Veldjes aanmaken

```
public class LijnPaneel extends JPanel implements ActionListener
{
    private JTextField afstandVeld;
    private JButton tekenKnop;
    private int afstand = -1;

    public LijnPaneel() {
        afstandVeld = new JTextField(4);
        tekenKnop = new JButton("Teken de lijnen");
        tekenKnop.addActionListener(this);

        this.add(new JLabel("Afstand tussen de lijnen"));
        this.add(afstandVeld);
        this.add(tekenKnop);
    }

    public void actionPerformed(ActionEvent e) { … }

    public void paintComponent(Graphics g) { … }
```

Fouten afhandelen

Er kan veel mislopen in een programma, sommige dingen kan de programmeur niet altijd voorzien:

- Foute gebruikersinput
- Bestand niet beschikbaar of beschadigd
- Geheugen is op
- enz…

Manueel afhandelen van fouten is onpraktisch: deze kunnen overal in de code voorkomen.

Geen optie:

Fouten Afhandelen

```
try {
    //Beschermde code
} catch(ExceptionType1 ex1) {
    //Catch blok
} catch(ExceptionType2 ex2) {
    //Catch blok
} catch(ExceptionType3 ex3) {
    //Catch blok
} finally {
        // het “finally”-blok wordt altijd
        // uitgevoerd, ook na een return statement
}
```

“finally” block is optioneel

Acties

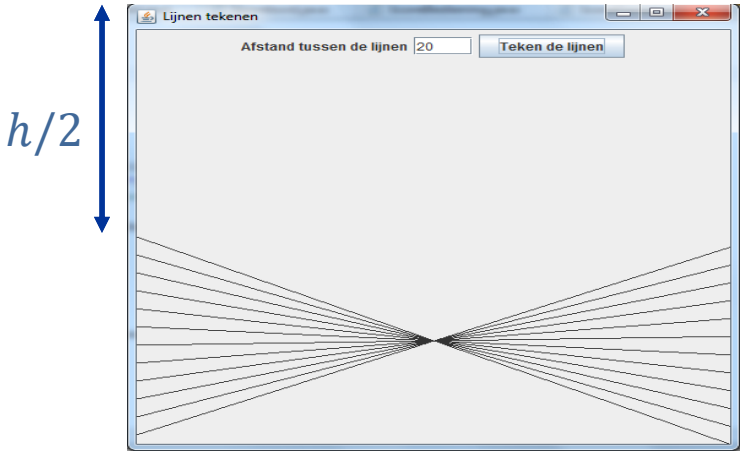
```
public void actionPerformed(ActionEvent e) {

try {
    int getal = Integer.parseInt(getalVeld.getText());
    if (getal <= 0)
                throw new ArithmeticException();
    repaint();
    }
    catch(NumberFormatException ex) {
    JOptionPane.showMessageDialog(null, "De input moet een getal zijn",
    "Foute ingave", JOptionPane.ERROR_MESSAGE);
    }
    catch(ArithmeticException ex) {
    JOptionPane.showMessageDialog(null, "Het getal moet groter dan 0 zijn",
    "Foute ingave", JOptionPane.ERROR_MESSAGE);
    }

}
```

Op dit punt wordt **paintComponent** (onrechtstreeks) opgeroepen

Lijnwaaier



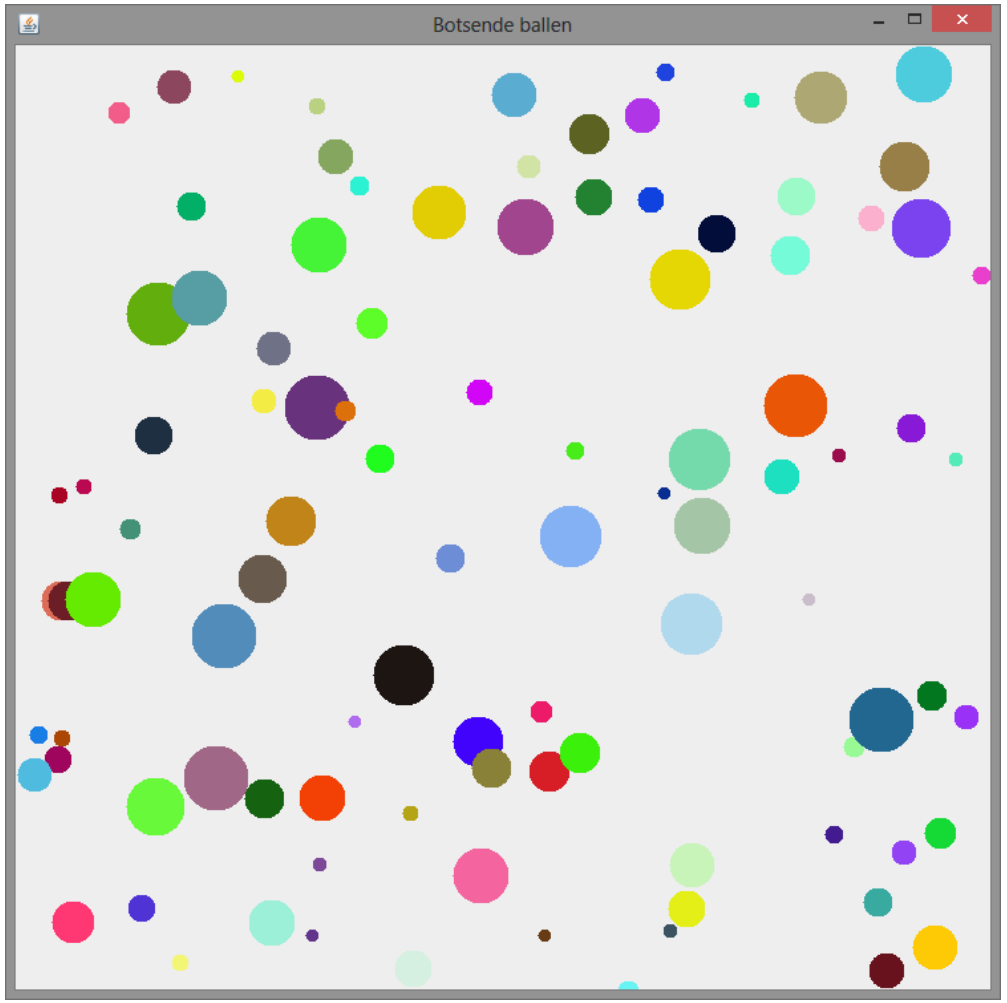

```
import java.awt.Graphics;


…
// Voorbeeld:
public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawLine(0, 0, this.getWidth(), this.getHeight());
}

public void actionPerformed(ActionEvent e) {
        …
        repaint();
}
```

Vraag 2: Botsende Ballen

- **Opgave:** maak een animatie met botsende ballen.
- **Deel 1:** teken alle ballen (statisch) op het scherm
- **Deel 2:** animaties voorzien



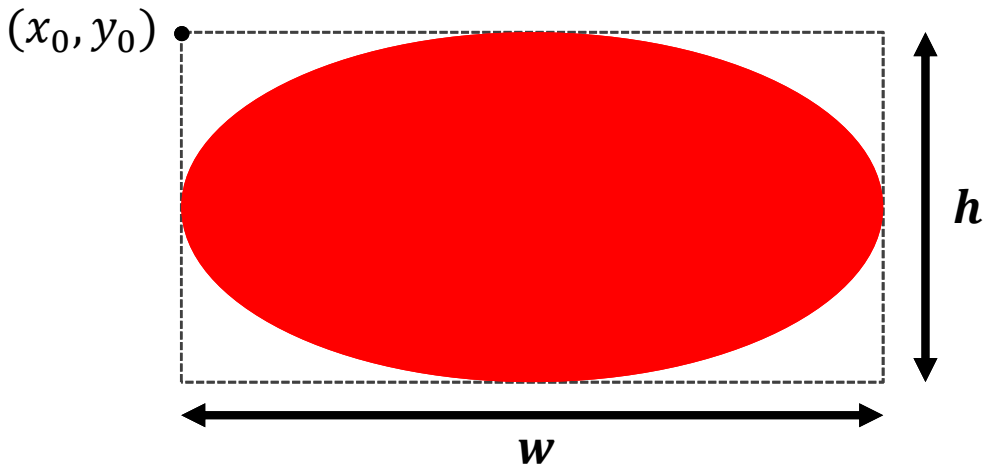
Vraag 2: Deel 1

Color: stelt kleur voor (met RGB coordinaten)

- `Constructor: Color(int R, int G, int B)`
  `// geef waarde tussen 0 en 255 mee`

Graphics: methodes

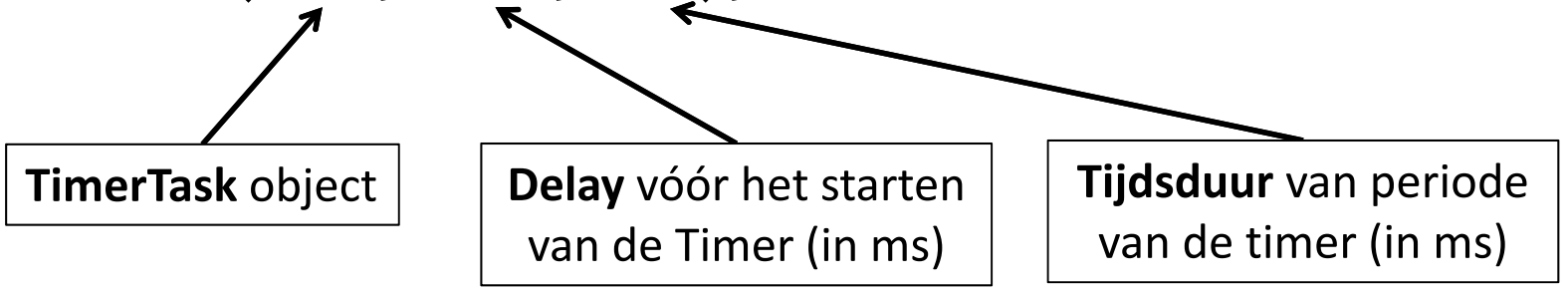
- `setColor(Color c) // kies kleur van “penseel”`
- `fillOval(int x0, int y0, int w, int h)`



Vraag 2: Deel 2

Klasse geschikt voor (periodisch) getimede fenomenen.

```
timer = new Timer();
timer.schedule(task, 1000, 200);
```




```
timer.cancel(); // stopt de Timer
```

TimerTask object beschrijft wat er periodisch moet gebeuren.

TimerTask

TimerTask bevat een methode **run()**, die beschrijft wat er gebeurt na elke periode van de **Timer**.

We willen een TimerTask met een eigen implementatie van **run()** → gebruik van een extensie (analoog met JPanel).

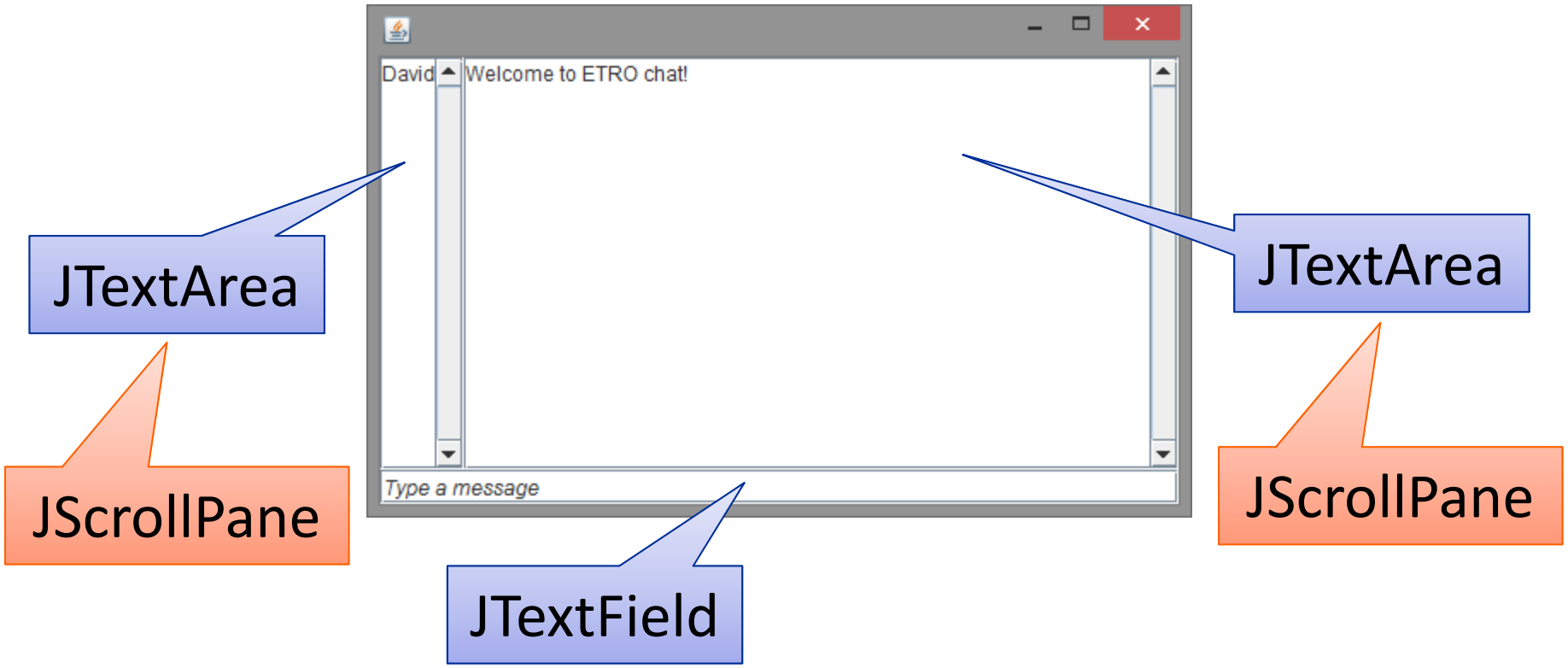
Typische implementatie:

```
public class MyTimerTask extends TimerTask {
        @Override
        public void run() { … }
}
```

We gebruiken de klasse lokaal enkel in **BalPaneel**. Declareer de klasse daarom in BalPaneel zelf (niet in een apart bestand).

Vraag 3: Chatclient

Opgave: een chatclient maken en verbinden op het netwerk.



Vraag 3: Chatclient

JScrollPane(Component view, int vsbPolicy, int hsbPolicy)

- view: onderdeel dat wordt vervat met een scrollbar
- vsbPolicy/hsbPolicy: wanneer de scrollbar moet verschijnen. Voor **vertical**:
  - JScrollPane.VERTICAL_SCROLLBAR_ALWAYS
  - JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED
  - JScrollPane.VERTICAL_NEVER

  (idem voor **horizontal**)

setLayout(*Layout object*): wat er moet gebeuren wanneer er componenten toegevoegd worden met **add()**;

- GridLayout, BoxLayout, SpringLayout, **Borderlayout**
- add(obj, BorderLayout.WEST);

SimpleConnection

Importeer de JAR-file

```
SimpleConnection connection = new SimpleConnection();

public void connect(String host, int port)
        // verbindt naar een server (zie bord voor inputs)
public void sendText(String msg);
        // stuur een bericht naar de server
public String readline();
        // kijkt of de server een bericht stuurt en returnt dat als String
```

SimpleConnection

SimpleConnection → **String readline()** kan de volgende berichten teruggeven:

- **SAY <naam> <bericht>**
  - betekent dat er iets gezegd werd
- **ENTER <naam>**
  - betekent dat er een nieuwe gebruiker in de chatroom is binnengekomen
- **LEAVE <naam>**
  - betekent dat er een gebruiker de chatroom heeft verlaten
- **USERS <naam 1>,<naam 2>,…,<naam N>**
  - geeft een lijst van de gebruikers die online zijn.

Tip: gebruik de volgende functies van String:

- **String[]** split(**String** splitter)
- **String** substring(**int** beginIndex)
- **String** substring(**int** beginIndex, **int** endIndex)